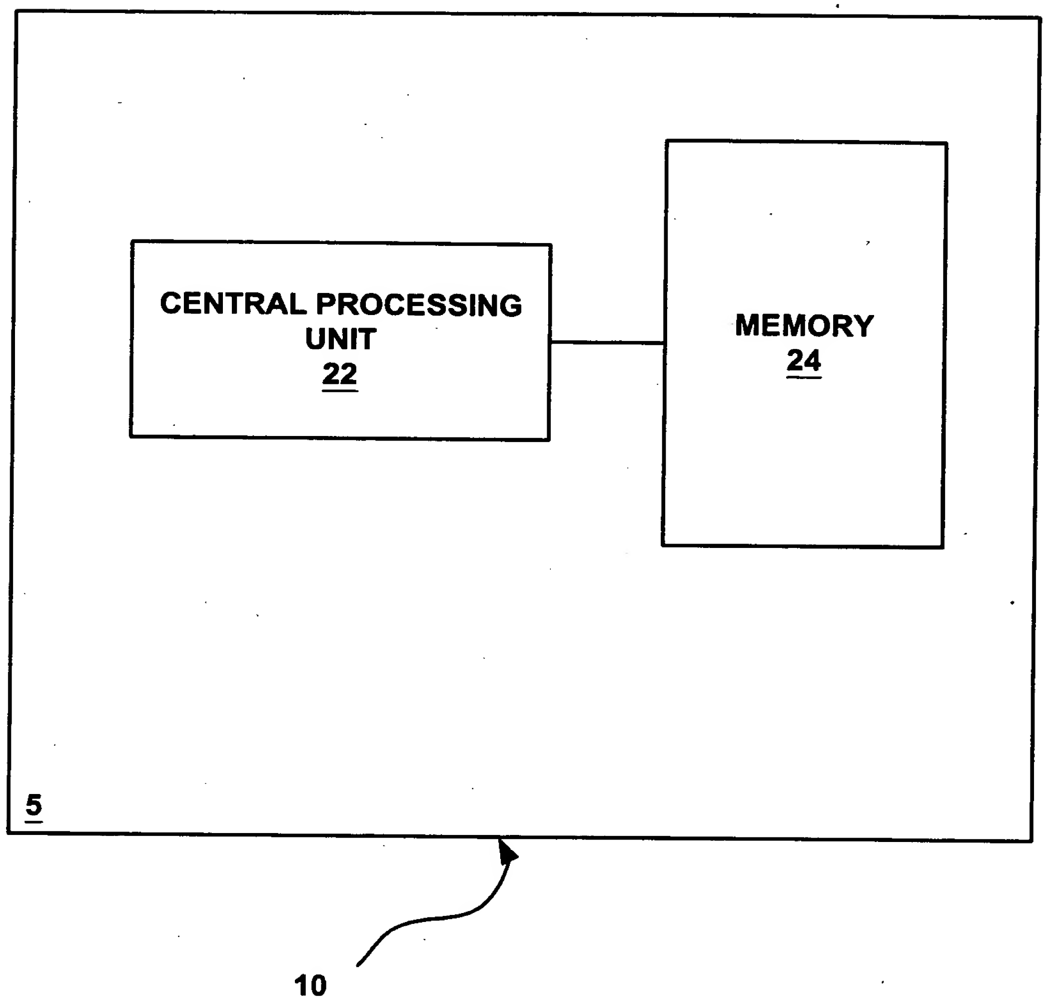
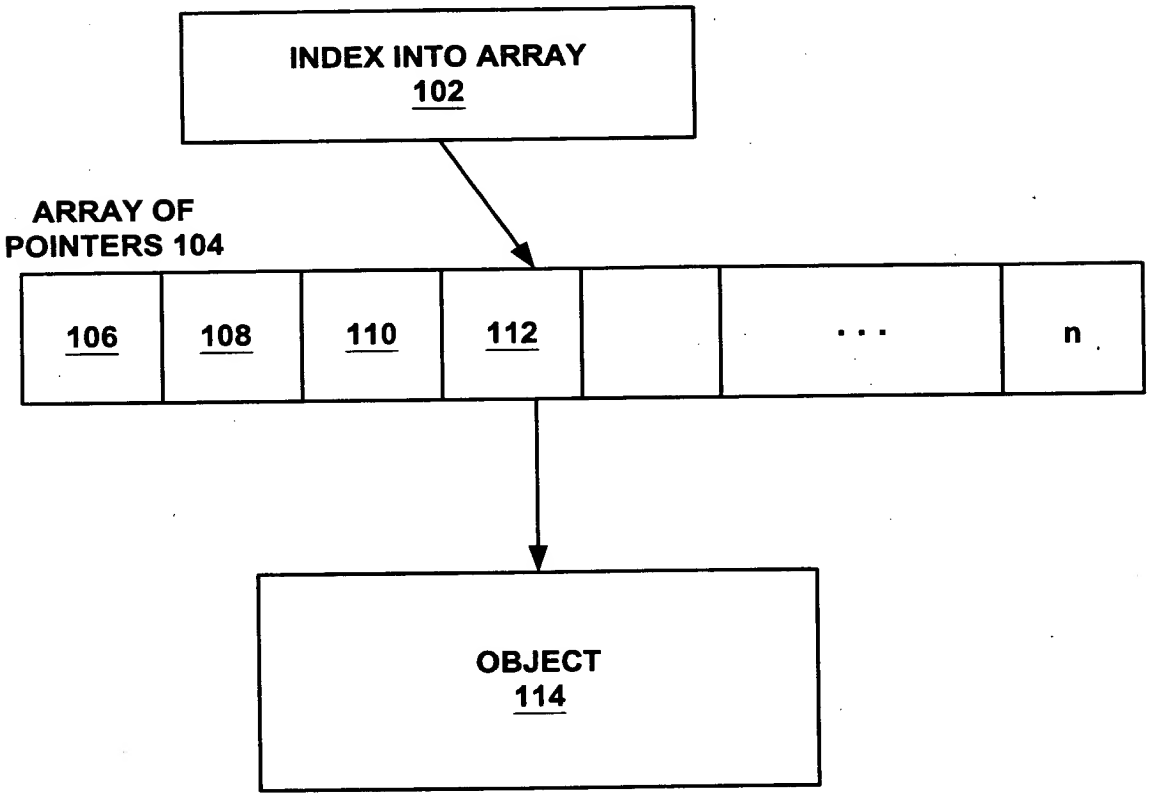


Figure 1



PRIOR ART

Figure 2

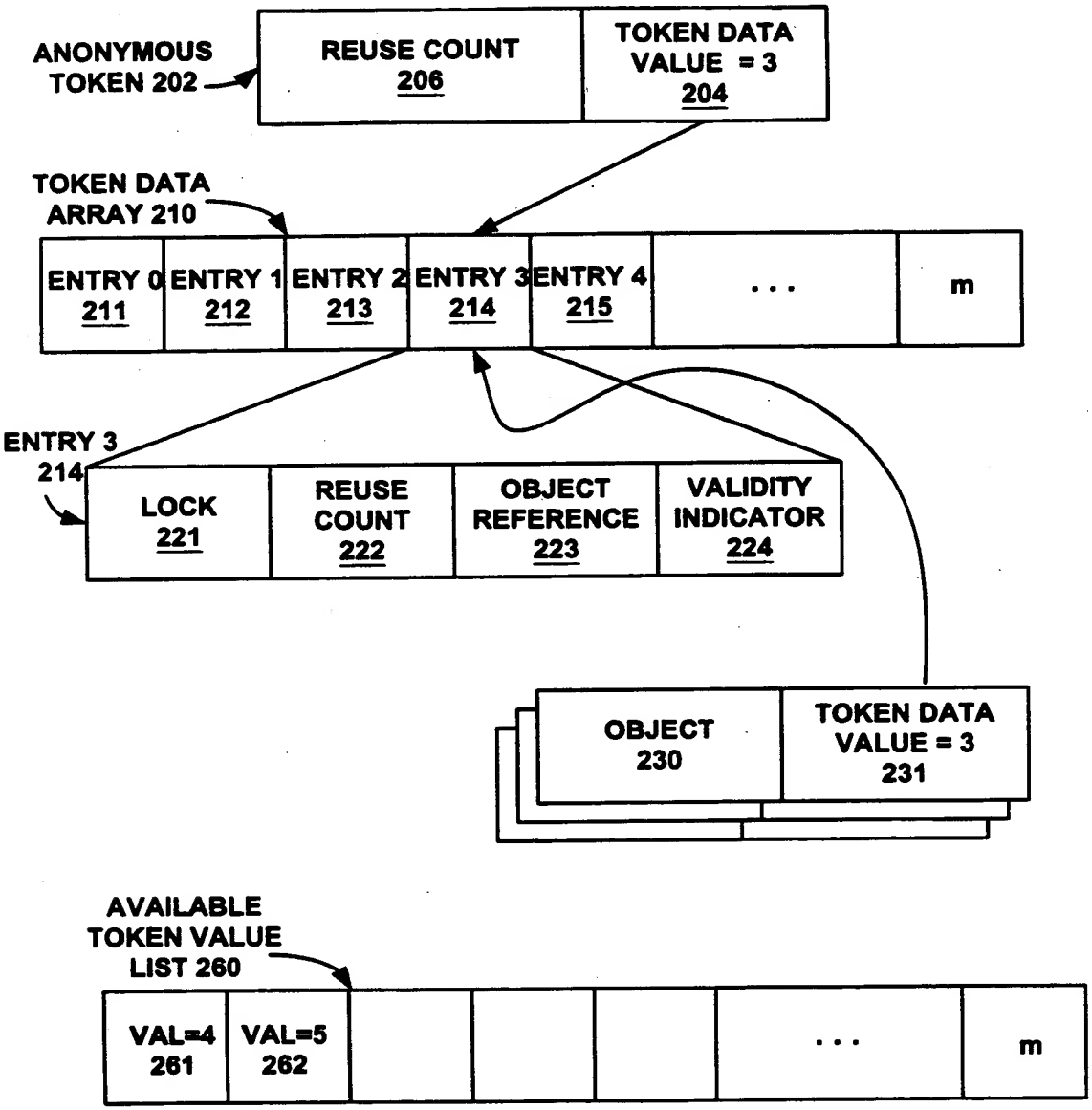

ANONYMOUS TOKEN 202
REUSE COUNT 206
TOKEN DATA VALUE = 3 204
TOKEN DATA ARRAY 210
ENTRY 0 211
ENTRY 1 212
ENTRY 2 213
ENTRY 3 214
ENTRY 4 215
. . .
m
ENTRY 3 214
LOCK 221
REUSE COUNT 222
OBJECT REFERENCE 223
VALIDITY INDICATOR 224
OBJECT 230
TOKEN DATA VALUE = 3 231
AVAILABLE TOKEN VALUE LIST 260
VAL=4 261
VAL=5 262
. . .
m

Figure 3

```
INTEGER PROCEDURE GET_TOKEN(REFERENCE TO OBJECT, REFERENCE TO TOKEN);

TOKENINXV := TOKEN_GETTOKEN;
  % TOKEN_GETTOKEN is a procedure which obtains an available slot
  % in the token data array.
401 % TOKEN_RETURNTOKEN is the complementary procedure.  It returns a
  % slot in the token data array to the available list.
  IF TOKENINXV EQL 0 THEN
  BEGIN
    % No slot available
402 GET_TOKENV := TOKEN_RESOURCELIMIT;
  END ELSE
  BEGIN
403 LOCK(TOKEN_DATA[TOKENINXV].LOCK);
    IF OBJECT.TOKENINX NEQ 0 THEN
    BEGIN
404   GET_TOKENV := TOKEN_ALREADYDONE;
      EXISTING_TOKENINXV := OBJECT.TOKENINX;
    END ELSE
    BEGIN
      TOKEN_DATA[TOKENINXV].REUSE_COUNT :=
       TOKEN_DATA[TOKENINXV].REUSE_COUNT+1;
      TOKEN_DATA[TOKENINXV].OBJECT_REFERENCE :=
405    REFERENCE TO OBJECT;
      TOKEN_DATA[TOKENINXV].VALID := TRUE;
      TOKENV.TOKEN_REUSECOUNTF :=
       TOKEN_DATA[TOKENINXV].REUSE_COUNT;
      TOKENV.TOKEN_INXF := TOKENINXV;
      OBJECT.TOKENINX := TOKENINXV;
      GET_TOKENV := TOKEN_SUCCESS;
    END;
406 UNLOCK(TOKEN_DATA[TOKENINXV].LOCK);
    IF GET_TOKENV EQL TOKEN_SUCCESS THEN
407 BEGIN
      TOKEN := TOKENV;
    END ELSE
    IF GET_TOKENV EQL TOKEN_ALREADYDONE THEN
    BEGIN
      LOCK(TOKEN_DATA[EXISTING_TOKENINXV].LOCK);
      IF TOKEN_DATA[EXISTING_TOKENINXV].VALID AND
408     (TOKEN_DATA[EXISTING_TOKENINXV].OBJECT_REFERENCE EQL
        REFERENCE TO OBJECT) THEN
      BEGIN
        TOKENV.TOKEN_REUSECOUNTF :=
         TOKEN_DATA[EXISTING_TOKENINXV].REUSE_COUNT;
        TOKENV.TOKEN_INXF := EXISTING_TOKENINXV;
      END ELSE
      BEGIN
409     GET_TOKENV := TOKEN_BADOBJECT;
      END;
      UNLOCK(TOKEN_DATA[EXISTING_TOKENINXV].LOCK);
410   TOKEN_RETURNTOKEN(TOKENINXV);
      IF GET_TOKENV EQL TOKEN_ALREADYDONE THEN
        TOKEN := TOKENV;
    END ELSE
    BEGIN
      TOKEN_RETURNTOKEN(TOKENINXV);
    END;
  END;
```

Figure 4

```
        INTEGER PROCEDURE
OPERATION_VIA_TOKEN(TOKEN,OPERATION_PARAMETERS);
        BEGIN
          INTEGER
            TOKENINXV;

          TOKENINXV := TOKEN.TOKEN_INXF;
          IF TOKENINXV LSS 1 OR
            TOKENINXV GEQ SIZE(TOKEN_DATA) THEN
420       BEGIN
            OPERATION_VIA_TOKEN := TOKEN_TOKENINVALID;
          END ELSE
          BEGIN
421       LOCK(TOKEN_DATA[TOKENINXV].LOCK);
            IF (NOT TOKEN_DATA[TOKENINXV].VALID) OR
              (TOKEN_DATA[TOKENINXV].REUSE_COUNT NEQ
               TOKEN.TOKEN_REUSECOUNTF) THEN
422         BEGIN
              OPERATION_VIA_TOKEN := TOKEN_TOKENINVALID;
            END ELSE
            BEGIN
              < perform operation on
423           TOKEN_DATA[TOKENINXV].OBJECT_REFERENCE,
              using OPERATION_PARAMETERS >
              OPERATION_VIA_TOKEN := TOKEN_SUCCESS;
            END;
424       UNLOCK(TOKEN_DATA[TOKENINXV].LOCK);
          END;
        END OPERATION_VIA_TOKEN;
```

Figure 5

```
                INTEGER PROCEDURE RETURN_TOKEN(REFERENCE TO
OBJECT);
                BEGIN
                  INTEGER
                    RETURN_TOKENV,
                    TOKENINXV;

              430   TOKENINXV := OBJECT.TOKENINX;
                    IF TOKENINXV EQL 0 THEN
                    BEGIN
                      % This assumes that OBJECT was allocated with TOKENINX
                      % set to zero.
                      RETURN_TOKENV := TOKEN_NOTOKEN;
                    END ELSE
                    BEGIN
                  431 LOCK(TOKEN_DATA[TOKENINXV].LOCK);
                      IF TOKEN_DATA[TOKENINXV].VALID CAND
                        (TOKEN_DATA[TOKENINXV].OBJECT_REFERENCE
EQL
                  432     REFERENCE TO OBJECT) THEN
                      BEGIN
                        TOKEN_DATA[TOKENINXV].VALID := FALSE;
                        TOKEN_DATA[TOKENINXV].OBJECT_REFERENCE :=
0;
                        OBJECT.TOKENINX := 0;
                        RETURN_TOKENV := TOKEN_SUCCESS;
                      END ELSE
                      BEGIN
                        RETURN_TOKENV := TOKEN_NOTOKEN;
                      END;
                  433 UNLOCK(TOKEN_DATA[TOKENINXV].LOCK);
                  434 IF RETURN_TOKENV EQL TOKEN_SUCCESS THEN
                        TOKEN_RETURNTOKEN(TOKENINXV);
                    END;
                  END;
              435 RETURN_TOKEN := RETURN_TOKENV;
END RETURN_TOKEN;
```

Figure 6

700

```
INTEGER PROCEDURE TOKEN_GETTOKEN;
BEGIN
440   LOCK(TOKEN_AVAILLOCK);
      IF TOKEN_AVAILHEAD EQL 0 AND
        SIZE(TOKEN_DATA) LSS TOKEN_MAXTOKENS THEN
      BEGIN
441     % TOKEN_MAXTOKENS is one more than the largest value
which
        % will fit in TOKEN_INXF and in TOKENINX of OBJECT,
assuming
        % that TOKEN_DATA is indexed origin zero.
        <Expand TOKEN_DATA, initializing new entries with
        LOCK free, VALID false, and REUSE_COUNT zero>
        < link new entries to avail list>
      END;
      IF TOKEN_AVAILHEAD NEQ 0 THEN
442   BEGIN
        <De-link and return head of avail list>
      END ELSE
      BEGIN
        TOKEN_GETTOKEN := 0;
      END;
443   UNLOCK(TOKEN_AVAILLOCK);
    END TOKEN_GETTOKEN;
```

701

```
    PROCEDURE TOKEN_RETURNTOKEN(TOKENINXV);
    VALUE   TOKENINXV;
    INTEGER TOKENINXV;
    BEGIN
      IF TOKEN_DATA[TOKENINXV].REUSE_COUNT LSS
TOKEN_REUSECOUNTMAX THEN
      BEGIN
450     % TOKEN_REUSECOUNTMAX is the largest value which will
fit in
        % TOKEN_REUSECOUNTF and in REUSE_COUNT of
TOKEN_DATA.
        LOCK(TOKEN_AVAILLOCK);
        < link to avail list>
        UNLOCK(TOKEN_AVAILLOCK);
      END;
    END TOKEN_RETURNTOKEN;
```

Figure 7